Distributed Shared Key Generation and Management Using Fractional Keys

Field of the Invention

The invention described herein pertains to communications, and more particularly to information security.

Related Art

Cryptographic key generation and management is an important problem in multicast and group communications (R. Canetti and Pinkas, B., "A taxonomy of multicast security issues," in *Internet-Draft* (1998); Harney, H. and Muckenhirn, C., "GKMP Architecture," *RFC 2093* (1997); Harney, H. and Muckenhirn, C., "GKMP Architecture," *RFC 2094* (1997); Ballardie, A., "Scalable Multicast Key Distribution," *RFC 1949* (1996); Poovendran, R., *et al.*, "A Scalable Extension of Group Key Management Protocol," *Proc. 2nd Ann.*

ATIRP Conf., Maryland, pp. 187-191 (1998), incorporated herein by reference). In many instances, it is desirable to generate a group *shared key* (SK) for efficient intra-group communications. However, having the same SK implies that all the group membership is at the same trust level. In a distributed, multicast group, it is often not possible nor desirable to have the *same* trust level throughout the group. One may be tempted to suggest that a single trust level can be defined by choosing the lowest possible trust level as the group trust level. Though such a straightforward approach is feasible, one can do better by compartmentalizing the group based on local trust levels (*Id.*). Such a compartmentalization inevitably least to *clustering* of a given group. Compartmentalization also helps in having a better control over the set of key management and distribution functions as noted in (*Id.*).

While the entities in each cluster may share a common trust level, it may be that the clusters are mutually suspicious and have only *partial* trust in each other. Thus, a mechanism is desired that permits mutually suspicious parties to come together to generate a shared key. In order to avoid involving (and potentially paying) a third party, it is also desirable that the scheme involve only the group members and not external parties.

Some schemes (such as Harney, H. and Muckenhirn, C., "GKMP Architecture," *RFC 2093* (1997); Harney, H. and Muckenhirn, C., "GKMP Architecture," *RFC 2094* (1997); Ballardie, A., "Scalable Multicast Key Distribution," *RFC 1949* (1996)) propose to replace the traditional (external) Key Distribution Center (KDC) with a *Group Controller* (GC) which can generate and distribute the keys. However, in these approaches, a single member is allowed to generate the keys. This means that group members must place complete trust in this group member. In (Poovendran, R., *et al.*, "A Scalable Extension of Group Key Management Protocol," *Proc. 2nd Ann. ATIRP Conf.*, Maryland, pp. 187-191 (1998)), a *panel* of members are allowed to generate the keys. However, this reference does not present any explicit distributed key generation scheme.

(Note: The following references are incorporated herein by reference: Bellare and Micali, "Non-Interactive Oblivious Transfer and Applications," in *Advances in Cryptology -- Crypto '89*, Springer-Verlag (1989), pp. 547-557; Poovendran *et al.*, "A Distributed Shared Key Generation Procedure Using Fractional Keys," *Proceedings of the MILCOM '98*, Boston, MA (Oct. 1998); Simmons, G.J., "An Introduction to Shared Secret and/or Shared Control Schemes and Their Applications," in *Contemporary Cryptology: The Science of Information Integrity*, Simmons, G.J., ed., IEEE Press (1992), pp. 441-497.)

Summary of the Invention

The invention described herein represents a new class of distributed key generation and recovery methods suitable for group communication systems where the group membership must be tightly controlled. The key generation approach allows entities which may have only partial trust in each other to jointly generate a shared key without the aid of an external third party. The group collectively generates and maintains a dynamic group binding parameter, and the shared key is generated using a pseudorandom function using this parameter as a seed. The methods employ distributed algorithms based on fractional keys (FK). The methods allow the members to automatically update the keys in a periodic manner without any assistance from an external third party, and to do so using verifiable secret sharing techniques. The key retrieval method does not require the keys to be stored in an external retrieval center. Note that many Internet-based applications may have these requirements.

Features and Advantages

The invention described herein has the feature of developing a shared key based on components associated with respective members of a cluster. The invention has the additional feature of a dynamic group binding parameter that

serves a seed for development of the shared key. The invention has the advantage of allowing cooperative key generation without requiring action by an independent party. The invention has the further advantage of allowing key retrieval without requiring the archiving of keys at an external retrieval center.

Brief Description of the Figures

The foregoing and other features and advantages of the invention will be apparent from the following, more particular description of a preferred embodiment of the invention, as illustrated in the accompanying drawings.

FIG. 1 is a flowchart illustrating the overall operation of an embodiment of the invention.

FIG. 2 is an example system implementing the invention.

FIG. 3 is a flowchart illustrating the initialization process as performed by a security manager, according to an embodiment of the invention.

FIG. 4 is a flowchart illustrating the initialization process as performed by cluster members in a distributed fashion, according to an embodiment of the invention.

FIG. 5 is a flowchart illustrating subsequent key generation, according to an embodiment of the invention.

FIG. 6 is a flowchart illustrating subsequent key generation using ElGamal public key pairs, according to an embodiment of the invention.

FIG. 7 is a flowchart illustrating key recovery, according to an embodiment of the invention.

FIG. 8 is a flowchart illustrating verification of security manager-based initialization, according to an embodiment of the invention.

FIG. 9 is a flowchart illustrating verification of distributed initialization, according to an embodiment of the invention.

FIG. 10 illustrates an example computing environment of the invention.

Detailed Description of the Preferred Embodiments

A preferred embodiment of the present invention is now described with reference to the figures where like reference numbers indicate identical or functionally similar elements. Also in the figures, the left most digit of each reference number corresponds to the figure in which the reference number is first used. While specific configurations and arrangements are discussed, it should be understood that this is done for illustrative purposes only. A person skilled in the relevant art will recognize that other configurations and arrangements can be used without departing from the spirit and scope of the invention. It will be apparent to a person skilled in the relevant art that this invention can also be employed in a variety of other devices and applications.

I. Properties of the Key Generation and Management Method

The following notation is used to describe the different entities involved in the method:

$\alpha_{i,j}$: The one-time pad of the *i*th member at the *j*th key update iteration.

θj: The group binding parameter at the *j*th key update iteration.

$\{K_i, K_i^{-1}\}$: Public key pair of the member *i*. This pair is assumed to be updated appropriately to preserve the integrity and confidentiality of any communication transaction by and with member *i*.

$FK_{i,j}$: The FK of the *i*th member at the *j*th key update iteration.

$HFK_{i,j}$: The *hidden* FK (HFK) of the *i*th member at the *j*th key update iteration.

SK_j: The group SK at the *j*th key update instance.

$A \rightarrow B:X$: Principal A sends principal B a message X.

In an embodiment of the invention, the message format is $\left\{\left\{T_i, M, j, Msg\right\}_{K_s^{-1}}\right\}_{K_R}$, where the variables are defined as follows:

- T_i: a real-valued, wallclock time stamp generated by member i
- M: denotes the *mode* of operation, with "I" for Initialization mode, "G" for Generation mode, and "R" for key Recovery mode.
- j: integer-valued, denotes the current iteration number.
- Msg: the message to be sent.
- K_s^{-1}: denotes the private key of the sender S.
- K_R: public key of the receiver R.

The following properties are desirable for a multiparty key generation scheme:

- An FK contributed by a participating member should have the same level of security as the group SK.
- A single participating member, without valid permissions, should not be able to obtain the FK of another member.
- If a FK-generating member has physically failed, been compromised or removed, the remaining FK-generating members should be able to jointly recover the FK of the failed member.

The first property simply states that the distributed key generation scheme has to be such that each FK space has at least the same size as the final SK space. Hence, each member may generate FK of different size but, when combined, they lead to a fixed length SK. The second property has to do with the need for protection of individual FKs that is desired in light of the absence of a centralized key generation scheme. In the current scheme, every member performs an operation to hide its FK such that, when all the hidden FKs (HFKs) and the group

parameter are combined, the net result is a new SK. Even if an HFK is known, the problem of obtaining the actual FK or the SK needs further computation. The requirements of the FK concealment mechanism are described in greater detail below.

If a contributing member physically fails, becomes compromised, or has to leave the multicast group, or cluster, then it becomes necessary to replace the existing member with a new member. Hence, the newly-elected member should be able to securely recover the FK generated by the replaced member. However, to ensure the integrity of the scheme, this recovery should be possible only if all the remaining contributing members cooperate. This feature deviates significantly from the existing key generating schemes (Harney, H. and Muckenhirn, C., "GKMP Architecture," *RFC 2093* (1997); Harney, H. and Muckenhirn, C., "GKMP Architecture," *RFC 2094* (1997); Ballardie, A., "Scalable Multicast Key Distribution," *RFC 1949* (1996)). The requirement that an individual member acting alone not be able to obtain the FKs of other contributing members is similar to protecting individual private keys in public key cryptography systems.

The following is a list of assumptions regarding the method:

- There exist two commutative operators ⊚ and ◇ which form an abelian group when operating on the set of keys.
- It is computationally difficult to perform cryptographic analysis on a cryptographically-secure random key by search methods if the key length is sufficiently large.
- The keys are all L bits in length, and all members know its length.
- The number of participants in generating the KS is fixed as n (where n may be a function of ⊚ and ◇).
- There is a mechanism for certifying the members participating in the key generation procedure, for securely exchanging the quantities required in the algorithm and for authenticating the source of these quantities.
- Every member has the capability to generate a cryptographically-secure random number of length L bits or longer.

With the assumptions above, the key management scheme can be described in terms of three major processes:

1. Initialization, which includes secure initial one-time pad and binding parameter generation and distribution;
2. Key Generation, an iterative process including fractional, hidden and shared-key generation; and
3. Key Retrieval, a process that is required only in the case of a member node failure or compromise.

These processes are collectively illustrated in process 100 of FIG. 1. Process 100 begins with a step 105. In a step 110, the key management process is initialized. Here, initial one-time pads are generated for each member. In addition, a binding parameter is generated and distributed to each member, permitting each member to generate the same key, a shared key SK. In a step 115, the members can operate securely using the SK. If, in a step 120, a failure occurs at a member's node, such as a compromise of the member or an equipment failure, then key retrieval is performed in a step 125. Here, recovery of the parameters associated with the failed node is performed. In a step 130, a new binding parameter is generated and new one-time pads are created. Operations then resume at step 115.

If, in step 120, no failure occurs, process 100 continues with a step 135. Here, a determination is made as to whether an update of the SK is needed. This may be required, for example, if a member leaves the cluster. Alternatively, an operation may simply require periodic updating of the SK. If an update is needed, key generation step 130 is performed. Operations then resume at step 115.

The processes of initialization, key generation, and key retrieval are described in greater detail below.

II. Initialization

A Group Initiator (GI) first selects a set of n FK-generating members of a cluster, and the GI may be one of these members. The GI can then contact a Security Manager (SM)–a third party who is *not* a FK-generating member–who generates the initial pads and the binding parameter and distributes them to the members. This is illustrated by system 200 of FIG. 2. Member 1, group initiator 210, is shown contacting security manager 250, who then distributes the necessary data to member 1 through 4, labelled 210 through 240, respectively. The data flow for this embodiment is illustrated by dotted lines. In an alternative embodiment, GI 210 initiates a distributed procedure among the group members (illustrated by solid lines) to create these quantities without the aid of an external party.

A. *SM-Based Initialization*

The process of initialization by an SM is illustrated in FIG. 3, process 300, according to an embodiment of the invention. Process 300 begins with a step 305. In a step 310, the GI generates an initial random one-time pad, $\alpha_{i,1}$, for each member i. In a step 315, an initial binding parameter θ_1 is computed such that $\alpha_{1,1} \circledcirc \alpha_{2,1} \circledcirc \ldots \circledcirc \alpha_{n,1} = \theta_1$. In steps 320 through 340, $\alpha_{i,1}$ and θ_1 are sent to each member i. In step 320, index i is initialized. In steps 325 and 330, the initial pads and binding parameter are distributed to member i, as

$$SM \rightarrow i: \left\{ \left\{ T_{SM}, I, 1, \alpha_{i,1}, \theta_1 \right\}_{K_{SM}^{-1}} \right\}_{K_i} .$$

In step 335, index i is incremented. In step 340, a determination is made as to whether $\alpha_{i,1}$ and θ_1 have been sent to all members i. If not, then $\alpha_{i,1}$ and θ_1 are sent to the next member i. The process concludes with a step 345. At the conclusion of process 300, each member has θ_1, on which a common SK can be based.

B. *Distributed Initialization*

In an alternative embodiment, initialization can be performed through a cooperative process involving all members, illustrated as process 400 of FIG. 4. The GI (assumed to be a member and denoted in process 400 by the index 1) can perform the following steps (see also Koblitz, N., *Cryptologia* 317-326 (1997), incorporated herein by reference) to generate the initial parameters of the group. Process 400 begins with a step 405. In a step 410, member 1 generates two uniformly-distributed random quantities γ and $v_{1,1}$ of bit length L. In a step 415, member 1 operates on these two quantities as $\gamma \circledcirc v_{1,1} = \delta_1$. In a step 420, member 1 sends the result to member 2 (the "next" member in the group) as $1 \rightarrow 2$: $\left\{\left\{T_1, I, 1, \delta_1\right\}_{K_1^{-1}}\right\}_{K_2}$.

Starting with member 2, each member i calculates its own δ_i based on the previous member's δ_{i-1}, and sends δ_i to the next member. This is illustrated in steps 425 through 450. In step 425, the index i is initialized to 2. In step 430, member i generates a uniform random variable $v_{i,1}$ of bit length L. In step 435, member i then operates on the quantity it received from member $i-1$ as $\delta_{i-1} \circledcirc v_{i,1} = \delta_i$. In step 440, member i then sends the result to member $i+1$ as $i \rightarrow i+1$: $\left\{\left\{T_i, I, 1, \delta_i\right\}_{K_i^{-1}}\right\}_{K_{i+1}}$.

In step 445, i is incremented. If, as determined in step 450, each of the n members has not generated a respective value δ_i, the process returns to step 430, where the next member i generates its uniform random variable $v_{i,1}$.

Eventually, the group member $i = n$ receives δ_{n-1} and, in a step 455, generates a uniformly-distributed random quantity $v_{n,1}$ of bit length L. In a step 460, member n performs $\delta_{n-1} \circledcirc v_{n,1} = \delta_n$. In a step 470, member n securely sends δ_n to the initiating member $i = 1$ as $n \rightarrow 1$: $\left\{\left\{T_n, I, 1, \delta_n\right\}_{K_n^{-1}}\right\}_{K_1}$. In a step 475,

the GI (member 1) then recovers δ_n and performs $\gamma \circledcirc \delta_n = \theta_1$. In steps 480 through 494, member 1 sends θ_1 to each member *i*. In step 480, the index *i* is initialized to 2. In step 485, member 1 sends θ_1 to member i as

$$1 \rightarrow i: \left\{\left\{T_1, I, 1, \theta_1\right\}_{K_1^{-1}}\right\}_{K_i}.$$

In step 490, each member *i* privately computes $\alpha_{i,1} = \theta_1 \circledcirc \nu_{i,1}$. In step 492, the index *i* is incremented. If, in step 494, i > n, so that each member *i* has received θ_1 and privately computed a respective $\alpha_{i,1}$, then the process 400 concludes with a step 496. Otherwise, the process returns to step 485, where member 1 sends θ_1 to another member. At the conclusion of process 400, each member has θ_1, on which a common SK can be based.

Note that these two approaches of initialization (security manager-controlled initialization and distributed initialization) are not equivalent unless additional security assumptions are made. For example, in the case of distributed initialization within the group, the following can be done.

Assume that members $i-1$ and $i+1$ conspire to obtain the secret member *i*, where the numerical ordering corresponds to the order of message passing in the distributed algorithm.

1. Member $i-1$ sends δ_{i-1} to member *i* as per the algorithm, and *also* to member $i+1$ without *i*'s knowledge.
2. Member *i*, who is unaware of the conspiracy between $i-1$ and $i+1$, computes $\delta_i = \delta_{i-1} \circledcirc \nu_{i,1}$ and sends it to member $i+1$ securely.
3. Member $i+1$ can now compute $\nu_{i,1} = \delta_{i-1} \circledcirc \delta_i$ and obtain the secret $\nu_{i,1}$ of member *i*.

However, the secret vi,1 generated by member *i* become part of the pads (i.e. the α's) of members $i-1$ and $i+1$. Hence, application of this initialization assumes that the parties are benign.

III. Key Generation

The key generation algorithm is an iterative process depicted in FIG. 5 as process 500. Each successive key generation, iteration j, requires as input a set of one-time pads $\alpha_{i,j}$, $i = 1, \ldots, n$, and the binding parameter θ_j, which are obtained from the initialization process (e.g., process 300 or process 400) for iteration $j = 1$, and from the preceding iterations for $j > 1$.

The iterative key generation process, according to an embodiment of the invention, consists of the following. Process 500 begins with a step 505. In steps 510 through 535, each member i generates a cryptographically-secure random number, fractional key $FK_{i,j}$, and sends it to every other member m. In step 510, index i is initialized to 1. In step 515, member i generates random number $Fk_{i,j}$. In step 520, member i generates a hidden fractional key $HFK_{i,j} = \alpha_{i,j} \circledcirc FK_{i,j}$. In step 525, member i sends $HFK_{i,j}$ to every other member m as

$$i \rightarrow m: \left\{ \left\{ T_i, G, j, HFK_{i,j} \right\}_{K_i^{-1}} \right\}_{K_m}.$$

In step 530, index i is incremented. If, as determined in step 535, each member i has created a respective $HFK_{i,j}$ and sent it to all other members, the process continues at a step 540. Otherwise, process 500 returns to step 515, where the next member i generates its respective $FK_{i,j}$.

Once the exchange of $HFK_{i,j}$'s is complete, each member computes the new group parameter θ_{j+1} and a new shared key SK_j. This occurs in steps 540 through 560. In step 540, index i is initialized to 1. In step 545, member i calculates the new binding parameter, $\theta_{j+1} = \lambda\theta_j \circledcirc HFK_{1,j} \circledcirc HFK_{2,j} \circledcirc \ldots \circledcirc HFK_{n,j} = FK_{1,j} \circledcirc FK_{2,j} \circledcirc \ldots FK_{n,j}$. In step 550, member i calculates a new one-time pad $\alpha_{i,j+1} = \theta_{j+1} \circledcirc FK_{i,j}$, and a new shared key $SK_j = f(\theta_{j+1})$ where $f(\cdot)$ is a strong one-way pseudo-random function. In step 555, index i is incremented. If, in step 560, $i > n$, so that each member i has created a new θ_{j+1} and a new SK_j, then the

process concludes with a step 565. Otherwise, process 500 returns to step 545, where the next member i calculates the new binding parameter, θ_{j+1}.

If the resulting group parameter θ_{j+1} is cryptographically insecure for a particular application, all members can repeat process 500 creating a new high quality group parameter θ_{j+1}.

At the end of process 500, we have the SK for the current iteration. Note that the quantity $\alpha_{i,j+1}$ is computed such that, for an outsider, obtaining $\alpha_{i,j+1}$ is very hard, even if the actual shared key SK_j is compromised at any key update time interval $(j, j+1)$. Knowing the shared key SK_j does not reveal the group parameter θ_j and, hence, the tight binding of the members will not be broken by the loss of the shared key.

Note the following additional features of the key scheme:

- Although all the members have each $HFK_{i,j}$, obtaining the $FK_{i,j}$ or $\alpha_{i,j+1}$ of another member involves search in the L-dimensional space, and obtaining their correct combination involves search in the $(n-1)L$ - dimensional space. Hence, even if a fellow member becomes an attacker, that rogue member faces nearly the same computational burden in obtaining the set of n FKs as an outside cryptographic analyst; i.e. trust is *not* unconditional.
- For such an outside attacker, breaking the system requires either search in an L-dimensional space to get θ, or nL - dimensional searches to break individual secrets of all the members. Access to all n HFKs is alone is insufficient to permit an attacker to determine the SK; for that, the attacker must also possess the current binding parameter θ which is time-varying and never transmitted. If an SK is known to be compromised (perhaps due to traffic analysis), information regarding θ is not obtained , since $f(\cdot)$ is a pseudo-random function.

In an embodiment of the invention, an $FK_{i,j}$ is used whereby $(FK_{i,j}^{-1}, FK_{i,j})$ is an individual ElGamal public key pair for the member i at update j. The iterative key generation process for this embodiment is illustrated as process 600 of FIG. 6. Process 600 begins with a step 605. In steps 610 through

640, each member i develops values $FK_{i,j}$ and $HFK_{i,j}$ and exchanges them with other members. In step 610, index i is initialized to 1. In step 615, member i randomly picks a number $FK_{i,j}^{-1}$ with $0 \le FK_{i,j}^{-i} \le p-2$. In step 620, member i generates $FK_{i,j} = \alpha^{FK_{i,j}^{-1}}$. Here, $(FK_{i,j}^{-1}, FK_{i,j})$ is an individual ElGamal public key pair for the member i at update j. In step 625, member i generates a quantity $HFK_{i,j} = (\alpha_{i,j} + FK_{i,j}) \bmod p$. In step 630, member i sends $FK_{i,j}$ and $HFK_{i,j}$ to each other member m, in the form

$$i \rightarrow m: \left\{ \left\{ T_i, G, j, HFK_{i,j}, FK_{i,j} \right\}_{FK_{i,j-1}^{-1}} \right\}_{FK_{m,j-1}}$$

. In step 635, index i is incremented. If, as determined in step 640, $i > n$, so that each member i has created a respective $HFK_{i,j}$ and sent it, along with $FK_{i,j}$, to all other members, the process continues at a step 645. Otherwise, process 600 returns to step 615, where the next member i selects its respective $FK_{i,j}^{-1}$.

In steps 645 through 665, each member generates a new binding parameter θ_{j+1} and one-time pad $\alpha_{i,j+1}$. In step 640, index i is initialized to 1. In step 650, each member i computes $\theta_{j+1} = \left((p-n-3)\theta_j + \sum_{i=1}^{i=n} HFK_{i,j} \right) \bmod (p-1)$, defining $GK_{j+1}^{-1} = \theta_{j+1}$. Each member i also computes $GK_{j+1} = \alpha^{GK_{j+1}^{-1}} = \prod_{i=1}^{i=n} FK_{i,j} = \prod_{i=1}^{i=n} \alpha^{FK_{i,j}^{-1}}$ in step 650. In step 655, member i calculates $a_{i,j+1} = (GK_j^{-1} + FK_{i,j}^{-1}) \bmod p$. In step 660, index i is incremented. In step 665, a determination is made as to whether $i > n$, i.e., whether each member i has calculated the new θ_{j+1} and a new $\alpha_{i,j+1}$. If so, process 600 concludes with a step 670. Otherwise, process 600 returns to step 650 so that the next member i can create a new θ_{j+1}.

Note that if the resulting group key pair $(GK_{j+1}, GK_{j+1}^{-1})$ is cryptographically insecure for a particular application, all members can repeat process 600, creating a new high quality key pair.

IV. Retrieval of the Fractional Key and One-time Pad of a Failed Node

The following steps, illustrated as process 700 of FIG. 7, are involved in recovery of the $FK_{i,j}$ and $\alpha_{i,j}$ of the node failed i, where j represents the iteration number in which the node was compromised or failed. The process begins with a step 705. In a step 710, any one FK-generating member–called the Recovery Initiator (RI)– initiates recovery and gives the HFK of the failed node i to the newly-elected node i as $RI \rightarrow i$: $\left\{\left\{T_{RI}, R, j, HFK_{i,j}\right\}_{K_{RI}^{-1}}\right\}_{K_i}$. In a step 615, the RI gives the newly-elected node i the current SK_j as $RI \rightarrow i$: $\left\{\left\{T_{RI}, R, j, SK_j\right\}_{K_{RI}^{-1}}\right\}_{K_i}$. In a step 720, distributed initialization is performed, with the following replacements: (a) θ by ξ and (b) $\alpha_{l,j}$ by $\beta_{l,j}$. Except for the changes in the notation and the number of members participating, the process for pad generation is same as for distributed initialization. Hence, at the end of this distributed pad generation, each member l has $\beta_{l,j}$ as its pad for key recovery process, and all these pads are bound with the parameter ξ. In steps 725 through 745, each member l calculates a modified hidden fractional key $\hat{HFK}_{i,j}$ and distributes it to newly elected member i. In step 725, index l is initialized to 1. In step 730, member l computes modified hidden fractional key $\hat{HFK}_{l,j} = \beta_{l,j} \diamond FK_{l,j}$ and sends it to the newly-elected member i as $l \rightarrow i$: $\left\{\left\{T_l, R, j, \hat{HFK}_{l,j}\right\}_{K_l^{-1}}\right\}_{K_i}$ in step 735. In step 740, index l is incremented. In step 745, a determination is made as to whether $l > n$, i.e.,

whether each member l has calculated a modified hidden fractional key $H\hat{F}K_{l,j}$ and distributed it to newly elected member i. If not, process 700 returns to step 730. Otherwise, process 700 continues with a step 750.

In step 750, member i combines all of the modified HFKs and recovers the fractional key $FK_{i,j}$ using the operation $FK_{i,j} = \lambda\xi \circledcirc H\hat{F}K_{i,j} \circledcirc \ldots \circledcirc H\hat{F}K_{i,j} \circledcirc \theta_{j+1}$. In step 755, member i extracts the one-time pad $\alpha_{i,j}$ using the operation $\alpha_{i,j} = HFK_{i,j} \circledcirc FK_{i,j}$. The process 700 concludes with a step 760.

Note that the recovered values of $FK_{i,j}$ and $\alpha_{i,j}$ are unique. Once the new node recovers the fractional key of the compromised node, it can inform the other contributing members to update the iteration number j to $j + 1$, and then all members can execute the key generation algorithm. Note that even though the newly-elected member recovers the compromised fractional key and pad, the next key generation operation of the new node does not use the compromised key or pad. Hence, even if the attacker possesses the fractional key or pad at iteration j, it does not allow the attacker to obtain the future fractional keys or pads without any computation.

V. A Specific Choice of the Functions ⊚ **and** ◇

A class of multiparty key generation algorithms is described above where a given instance of the class is determined by choice of function ⊚. Note that one possible choice for ⊚ is the modulo addition operation with respect to a large odd prime p, denoted here with ⊕. In this case, we can deduce the following computation from the key generation algorithm:

$$HFK_{1,j} \oplus HFK_{2,j} \oplus \cdots \oplus HFK_{n,j} = FK_{1,j} \oplus FK_{2,j} \oplus \cdots \oplus FK_{n,j} \oplus (n-1)\theta_j$$

To remove the effect of θ_j on θ_{j+1}, we should ensure that $\lambda = (p + 1 - n)$ so that

$$\begin{aligned}\theta_{j+1} &= (p+1-n)\theta_j \oplus HFK_{1,j} \oplus HFK_{2,j} \oplus \cdots \\ &\quad \cdots \oplus HFK_{n,j} \\ &= FK_{1,j} \oplus FK_{2,j} \oplus \cdots \oplus FK_{n,j}\end{aligned}$$

Regarding the choice of the number of members, clearly the choice of $n = 2$ is not appropriate for such a scheme. Although choosing $n = 3$ does not instantly expose a secret pad α; when a participating member becomes an attacker (i.e. a *rouge*), the following attack–called *fractional attack* (FA)–is feasible.

Lemma: When $\circledcirc$ is an $\oplus$ function, independent of how nontrivial the bit-length of the key is, choosing $n = 3$ permits a FA.

Proof: Assume that the time instant at which one member i ($i = 1$ or 2 or 3) become a *rogue* is j. At this time the member have values of $\alpha_{1,j} = HFK_{2,j} \oplus HFK_{3,j}$, $\alpha_{2,j} = HFK_{3,j} \oplus HFK_{1,j}$, $\alpha_{3,j} = HFK_{1,j} \oplus HFK_{2,j}$. Every member also has access to the current θ_{j+1} and their own $FK_{l,j}$ ($l = 1, 2, 3$). At this stage, obtaining the α component of any other member is as computationally intensive as an outside attacker trying to obtain θ_{j+1}. However, if a member, say $i = 1$, is compromised and releases its secret $\alpha_{1,j}$, then each of the other members can use this and compute $FK_{1,j} = \alpha_{1,j} \oplus \theta_j$. Since $\theta_{j+1} = FK_{1,j} \oplus FK2_{2,j} \oplus FK_{3,j}$, each member can now compute the other non-rogue member's FK as well.

This leads to the following corollary: When $\circledcirc$ is an $\oplus$ function, independent of how non-trivial the bit-length of the key, the minimum number of members to prevent a FA by a single *rogue* member for the multiparty key scheme is 4.

VI. Verifiable Secret Sharing

Since there are multiple entities involved in key generation, it becomes important to have a mechanism to verify if the parameters exchanged actually contribute to the generated shared key. The verification steps can be followed at (1) SM-based group initialization, (b) distributed group initialization, and (c) θ-generation iteration.

A. SM-based Initialization

In the case of the SM-based scheme, each member i needs to make sure that the SM uses non-trivial values of its $\alpha_{i,1}$ and θ_1. Since each member needs to protect its individual pad value, one method for openly checking correctness of the pads is to generate a public value that will enable all the key generating members to check their correctness without revealing the actual value of the individual pads. Such a verification technique falls under the category of Verifiable Secret Sharing (VSS) (Feldman, P., "A Practical Scheme for Non-Interactive Verifiable Secret Sharing," *Proc. of IEEE Fund. Comp. Sci.*, pp. 427-437 (1987); Pedersen, T. P., *Advances in Cryptology - CRYPTO, LNCS 576*:129-140 (1991)).

If one wants to check if the individual initial pads $\alpha_{i,1}$ given by the security manager are "good", process 800 of FIG. 8 can be used. The process begins with a step 805. In a step 810, one member (possibly the SM) picks a very large prime number q. The number picked should be larger than the possible range of the θ value. In a step 820, prime number q is sent to all the members. In a step 825, the same member also sends a generator g of the multiplicative group q. In a step 830, each member picks a random polynomial f_i having a value 0 at the origin. In a step 835, each member adds the polynomial to its pad value, generates $\hat{\alpha}_{i,1} = g^{\alpha_{i,1}+f_i}$ and broadcasts the values to all the members. In a step 840, each member i computes $g^{\theta_1} = \prod_{j=1}^{j=n} \hat{\alpha}_{i,1} = g^{\theta_1}$. In a step 845, each member

checks if the value is equal to g^{θ_1} at the origin. If not, then the verification fails in a step 850. If the check of step 845 passes, then in a step 855, each member checks to see that

$$g^{\alpha_{i,1}} = \frac{g^{\theta_1}}{\prod_{j=1}^{j=n} g^{\alpha_{j,1}}}$$

If not, verification fails in step 850. Failed verification means that some or all of the members' one-time pads do not correspond to θ_1. Process 800 concludes with a step 860.

B. Distributed Initialization

In the case of distributed initialization, process 900 of FIG. 9 can be used to check if the GI, member 1, has produced a θ_1 using contributions from all the group members. The process begins with a step 905. In a step 910, one member (possibly the GI) picks a very large prime number q. The number picked should be larger than the possible range of the θ_1 value. In a step 915, prime number q is sent to all the members. In a step 920, the same member also sends a generator g of the multiplicative group under q to all members. In a step 925, GI computes g^{γ} and $g^{v_{1,2}}$, and makes them available to all the group members. In a step 930, each member i publishes $g^{v_{i,1}}$ making it available only to the group members. In a step 935, each member i checks if $g^{\theta_1} = \prod_{j=1}^{j=n} g^{v_{j,1}}$. If the equality is not true, then failed verification is indicated in a step 940. Failure (inequality) means that the binding parameter θ_1 and the individual one-time pads do not agree. Since at each step of adding their secrets members published the broadcast values, it is possible to check which member cheated if there is no collaboration. If there

is a collaboration, then the last among the collaborating member can be identified by the non-collaborating member.

Note that similar testing can be done for the key generation process.

VII. Environment

The present invention may be implemented using hardware, software or a combination thereof. The operations described above may be implemented in a computer system or other processing system at the node of a member. An example of such a computer system 1000 is shown in FIG. 10. The computer system 1000 includes one or more processors, such as processor 1004. The processor 1004 is connected to a communication infrastructure 1006, such as a bus or network). Various software implementations are described in terms of this exemplary computer system. After reading this description, it will become apparent to a person skilled in the relevant art how to implement the invention using other computer systems and/or computer architectures.

Computer system 1000 also includes a main memory 1008, preferably random access memory (RAM), and may also include a secondary memory 1010. The secondary memory 1010 may include, for example, a hard disk drive 1012 and/or a removable storage drive 1014, representing a floppy disk drive, a magnetic tape drive, an optical disk drive, etc. The removable storage drive 1014 reads from and/or writes to a removable storage unit 1018 in a well known manner. Removable storage unit 1018, represents a floppy disk, magnetic tape, optical disk, or other storage medium which is read by and written to by removable storage drive 1014. As will be appreciated, the removable storage unit 1018 includes a computer usable storage medium having stored therein computer software and/or data.

In alternative implementations, secondary memory 1010 may include other means for allowing computer programs or other instructions to be loaded into computer system 1000. Such means may include, for example, a removable

storage unit 1022 and an interface 1020. Examples of such means may include a program cartridge and cartridge interface (such as that found in video game devices), a removable memory chip (such as an EPROM, or PROM) and associated socket, and other removable storage units 1022 and interfaces 1020 which allow software and data to be transferred from the removable storage unit 1022 to computer system 1000.

Computer system 1000 may also include a communications interface 1024. Communications interface 1024 allows software and data to be transferred between computer system 1000 and external devices. Examples of communications interface 1024 may include a modem, a network interface (such as an Ethernet card), a communications port, a PCMCIA slot and card, etc. Software and data transferred via communications interface 1024 are in the form of signals 1028 which may be electronic, electromagnetic, optical or other signals capable of being received by communications interface 1024. These signals 1028 are provided to communications interface 1024 via a communications path (i.e., channel) 1026. This channel 1026 carries signals 1028 and may be implemented using wire or cable, fiber optics, a phone line, a cellular phone link, an RF link and other communications channels. In an embodiment of the invention in which computer system 1000 represents the computer system of a member's node, signals 1028 comprise information flowing to and from the node, such as the encrypted form of δ_i in step 440, and the encrypted form of $HFK_{i,j}$ of step 525.

In this document, the terms "computer program medium" and "computer usable medium" are used to generally refer to media such as removable storage units 1018 and 1022, a hard disk installed in hard disk drive 1012, and signals 1028. These computer program products are means for providing software to computer system 1000.

Computer programs (also called computer control logic) are stored in main memory 1008 and/or secondary memory 1010. Computer programs may also be received via communications interface 1024. Such computer programs, when executed, enable the computer system 1000 to implement the present invention

as discussed herein. In particular, the computer programs, when executed, enable the processor 1004 to implement the present invention. Accordingly, such computer programs represent controllers of the computer system 1000. Where the invention is implemented using software, the software may be stored in a computer program product and loaded into computer system 1000 using removable storage drive 1014, hard drive 1012 or communications interface 1024. In an embodiment of the present invention, the steps of processes 300 through 900 are implemented in software that can therefore be made available to processor 1004 at a member node through any of these means.

VIII. Conclusion

While various embodiments of the present invention have been described above, it should be understood that they have been presented by way of example, and not limitation. It will be apparent to persons skilled in the relevant art that various changes in detail can be made therein without departing from the spirit and scope of the invention. Thus the present invention should not be limited by any of the above-described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.